超並列計算機 JUMP-1 のクラスタバスにおけるメモリアクセスのバッファリング

福島 直人

概　要

近年、プロセッサの速度向上は著しい。しかし、記憶装置へのアクセス速度はそれほど向上していない。このため、プロセッサの動作速度と記憶装置へのアクセス速度に大きな差が生じている。よって、この差を何らかの形で補償する必要がある。

この目的を実現する手法としてメモリアクセスと演算のオーバラップ、ならびにメモリアクセスのパイプライニングがある。前者はメモリアクセスとプロセッサの命令実行を非同期的に行うことで、メモリアクセスのレイテンシを見かけ上短縮する手法である。一方後者は、メモリアクセス経路を複数のステージに分割し、複数のメモリアクセスをパイプライン処理することでメモリアクセス経路のスループットを向上する手法である。

これらの手法を実現するためには、一時的にメモリアクセスに関する情報をバッファリングする必要がある。本報告では、現在我々が開発中の超並列計算機 JUMP-1 のプロセッサ・ユニットに実装したメモリアクセスのバッファリング技法について報告する。

JUMP-1 ではプロセッサとメモリの速度差を補償するもう一つの手法であるキャッシュを設けている。したがって、プロセッサとバスとの間にキャッシュ (データアレイとタグアレイ) とその制御回路 (キャッシュ・コントローラ) を設ける。JUMP-1 のプロセッサ・ユニットでのメモリアクセス・バッファリングはこのキャッシュ・コントローラ内で行う。つまり、キャッシュ・コントローラは通常のキャッシュ動作に関する制御と同時にバッファの管理を行う。さらに JUMP-1 ではバス上でのキャッシュ・コヒーレンス制御の高速化のためにキャッシュのタグアレイのコピー (バックマップと呼ぶ) を設けプロセッサとバスが同時にキャッシュを検索することを可能としているため、タグアレイとバックマップとの一貫性制御を行わなければならない。

一方、メモリアクセスのバッファリングを行うことで、システム内で一時的に一貫性が保たれない状態が発生する。したがって、キャッシュ・コントローラはこのような一時的な一貫性の欠如を回避あるいは補償するための制御も要求される。

本研究ではこのようなキャッシュ・コントローラをゲートアレイを使って設計・実装を行った。その結果バッファリングに要するゲート数は約 11,000 ゲートとなった。JUMP-1 にキャッシュを実装することにより、平均メモリアクセス時間は、しない場合と比較して 6.7% 程度に減少し、また、メモリアクセスと演算の並列実行を行うことで、2 次キャッシュ・アロケート・ライトの平均時間は 29% 短縮することが分かった。

Memory Access Buffering on the Cluster Bus of the Massively Parallel Computer JUMP-1

Naoto Fukushima

Abstract

Recently, clock-speed of processors is increasing quickly. But access time to memory devices decreases not so quickly. These facts cause great difference of speed between processors and memories. We must narrow this gap to improve system performance.

Overlapping memory accesses with arithmetic operations and Memory-Access-Pipelining are techniques to accomplish this purpose. The former is a technique to reduce memory access latency by executing memory accesses asynchronously with processor's executions. The latter is a technique to improve the throughput of memory access channels by separating themselves into several stages and pipelining memory accesses.

To realize these methods, informations about the memory access must be buffered in somewhere. In this report, we describe Memory-Access-Buffering technique implemented on the processor elements of JUMP-1, a massively parallel computer which we are developing.

In JUMP-1, cache memory system, which is another technique to narrow the gap between processor speed and memory speed, is implemented. A cache, which is composed of a data array and a tag array, and a control device which is called Cache Controller are placed between a processor and the cluster bus. Memory-Access-Buffering at the processor element in JUMP-1 is also managed by the Cache Controller. Moreover, in JUMP-1, the copies of cache tags, called Back Map, are implemented to speed up the cache coherence control. This makes possible to refer to cache simultaneously from both sides of Cache Controller (processor and bus). But the control mechanism to maintain coherence between a tag array and a Back Map is required on the Cache Controller.

On the other hand, the Memory-Access-Buffering technique causes temporary inconsistency in the computer system. Consequently, the control mechanisms to avoid or recover from this temporary inconsistency are also required on the Cache Controller.

In this research, we designed the Cache Controller with these mechanisms and implemented it with custom gate array LSIs. The results show that the number of gates for the Memory-Access-Buffering are about 11,000. And we also found that the memory access latency is reduced to 6.7 per cent as much by implementing the Cache Memory on JUMP-1, and that the average time of cache allocate writes is reduced by 29% when the mechanism for parallel execution of arithmetic operations and memory accesses is implemented.

1　はじめに

計算機の高速化の需要はとどまるところを知らない。計算機の構成要素として重要なものにプロセッサとメモリがあるが、プロセッサの動作速度は年々大きく向上しているのに対し、メモリへのアクセス速度はそれほど向上していない [1]。このため、両者の動作速度に大きな差が生じており、プロセッサにとって、メモリアクセスのレイテンシは非常に大きいものになっている。計算機の高速化のためにはこれを補償する必要がある。

メモリへのアクセス速度を向上する直接的な方法としては、メモリのデータ幅を広げ、1回のアクセスで転送できるデータの量を増やし、スループットを向上させる方法や、メモリをインタリーブ構成とし連続アドレスへのアクセス時のスループットを向上させる方法、また並列計算機においては、メモリを複数のバンクに分け、ことなるメモリバンクへの同時アクセスを可能にする方法などがある [1]。

しかし、システム規模の拡大とともに、このようなメモリ周辺の工夫だけでは十分な性能向上が得られなくなりつつある。したがって、プロセッサとメモリの間に介在するすべてのハードウェア構成要素を含めた、システム全体でメモリアクセスのレイテンシに対処する必要がある。

このようなシステムレベルでの対処法として、メモリアクセスと演算のオーバラップ、ならびにメモリアクセスのパイプライニングがあり、これらを実現するために必要なハードウェア技術がメモリアクセス・バッファリングである [9]。

また、キャッシュ・メモリの導入によるメモリアクセス・レイテンシの隠蔽は非常によく用いられる手法であるが、ここでもメモリアクセスのバッファリングがシステムの性能向上に寄与する。

本稿では、第2章でメモリアクセスの高速化手法について述べる。第3章で現在開発中の超並列計算機 JUMP-1 に対するこれらの高速化手法の実装について述べる。第4章で、実装結果に関する考察を行う。

2　メモリアクセスの高速化手法

2.1　用語の定義

本稿で用いる用語の定義を行う。

- デバイス
  ネットワークに接続されている処理ユニット。本稿の対象とする並列計算機においてはプロセッサ、キャッシュ・メモリ、キャッシュ・コントローラをまとめたプロセッサ・ユニットと、メモリとメモリコントローラをまとめたメモリ・ユニットがそれぞれデバイスである。
- パケット
  デバイス間で行われる情報伝達の最小単位。パケットには、アドレス、データ、および行うべき処理が記述されている。本稿においてはパケットには、要求を行う要求パケットと、要求に対する応答パケットの2種類ある。
- トランザクション
  データの移動、およびそれにともなうシステムの状態変化を指す。

2.2　メモリアクセス・バッファリング

2.2.1　メモリアクセスと演算のオーバラップ

プロセッサが発行するメモリアクセス命令のうちリード命令は、それに対応するメモリアクセスが行われ、プロセッサにデータが渡されるまでプロセッサの処理を中断しなければならない。これに対してライト命令は、当該ライト・アクセスが必ず実行されるという保証があれば、プロセッサは必ずしもそのライト・アクセスの完了を待つ必要がない。したがって、プロセッサの処理の実行と並行してメモリアクセスを実行するハードウェア機構を用意すると、プロセッサはその機構にメモリアクセスを依頼した直後に次の処理を開始することが可能となる。これにより、プロセッサから見たみかけ上のメモリレイテンシを短縮することができる。この目的で設けられるハードウェア機構がライト・バッファである。

図1にライト・バッファを設けたシングルプロセッサシステムの構成を示す。この図ではライト・アクセスはメモリではなくライト・バッファへ、リード・アクセスは直接メモリへ行われることを仮定している。

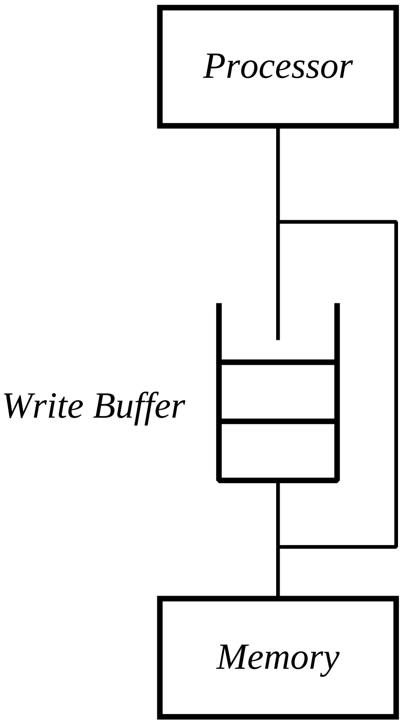


図1: ライト・バッファを設けたシングルプロセッサシステムの例

例えば、プロセッサの動作速度が100MHz（1サイクル10ns）、ネットワークにおける遅延が20ns、メモリアクセスに100ns かかるとする。メモリアクセス・バッファリングが行われない場合、メモリアクセスは図2(a)に示されるタイミングで行われる。ストアがプロセッサから発行されると、2サイクル後にメモリにおいてアクセスが開始する。その後10サイクルしてメモリへのデータ書き込みが終了して、そのまた2サイクル後に書き込み終了がプロセッサに伝えられる。この間プロセッサの動作が停止することになる。

しかし、プロセッサとメモリの間にライト・バッファを加えて、メモリアクセスをバッファリングした場合、次のように処理が行われる。プロセッサはデータをライト・バッファに書くだけでストアの実行が終了したことにして、動作を先に進める。一方、実際のメモリへの書き込みはライト・バッファが行う（図2(b)）。したがって、プロセッサの動作とメモリへのストアが同時に行われる。プロセッサがストアのために停止する時間は2サイクル（ライト・バッファへの書き込みに1サイクル、ライト・バッファからの Acknowledgement に1サイクル）ですみ、メモリアクセス・バッファリングを行わない場合（14サイクル）の7分の1で済む。プロセッサにとってはメモリアクセスのレイテンシは大幅に減少することになる。

2.2.2　メモリアクセス・パイプライニング

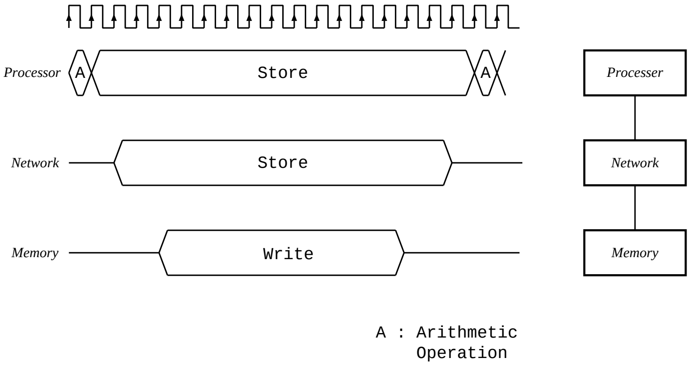
メモリアクセスのパイプライニングは、複数のメモリアクセスを同時に処理することで、メモリアクセスのスループットを向上する技術である。

図2(b)のメモリアクセスタイミングにおいて、ネットワーク上での情報の伝達について考える。この図では、プロセッサ（あるいはライト・バッファ）がメモリアクセスを要求し、メモリがそれに対して応答するまでの間、継続してバスが占有されている。しかしながら、この間、有効な情報の伝達が行われているの

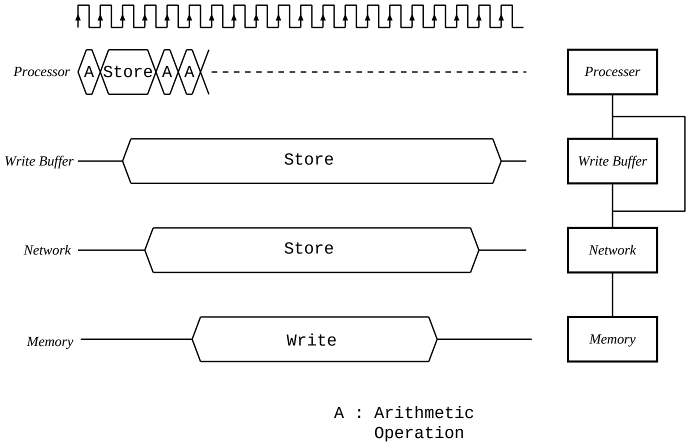
は、最初のアクセス要求の伝達と、最後の応答の伝達のみで、その間は何の情報伝達も行われていない。したがって、メモリアクセス要求とそれに対応する応答を切り離し、2つのフェーズに分けてメモリアクセスを実行すれば、不必要なバスの占有をなくし、ネットワークの利用効率を向上することが可能になる。このために必要となるハードウェアが送信バッファと受信バッファである。

2つのプロセッサ(Pa,Pb)と2つのメモリ(Ma,Mb)があるシステム（図3(a)）において、PaがMaに対するリードを、PaがMbに対するストアをほぼ同時に開始した場合を考える。この場合のメモリアクセスのタイミングを図3(b)に示す。Paからのメモリアクセスがキャッシュ・コントローラからネットワークを流れて、Maのメモリコントローラに入る。ここからMaに対して読み出しが行われる。この間にPbからのメモリアクセスがバスを流れて、Mbのメモリコントローラに入り、Mbに書き込みが行われる。その後、Maからデータが出力されて、一旦メモリコントローラで貯められ、その後ネットワークを通ってPaに渡る。またMbへの書き込みが終了し、Acknowledgementがネットワークを通ってPbに伝わる。

このように、Maへのアクセスと Mbへのアクセスが同時に実行されることで、バスのスループットが向上する。

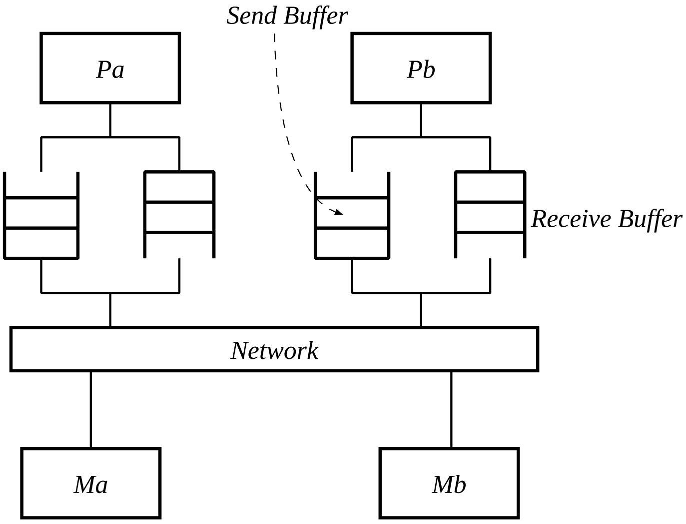


(a):ライト・バッファを設けない場合のストア・アクセスのタイミング

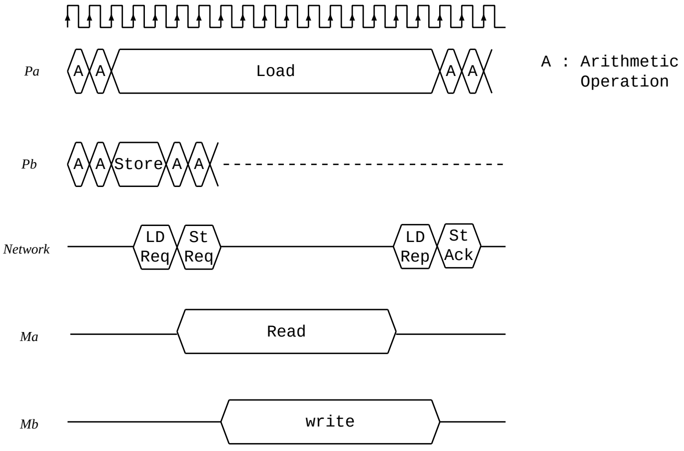


(b):ライト・バッファを設けた場合のストア・アクセスのタイミング

図2: ライト・バッファを用いたメモリアクセスのバッファリング効果



(a):ハードウェア構成



(b):アクセスタイミング

図3: メモリアクセス・パイプライニングによる処理の流れ

2.3 キャッシュメモリ

一般に、主記憶参照には局所性があり、命令やデータの主記憶参照はある小さな時間間隔で見れば主記憶の局所的な領域に集中している。したがって、この領域を容量は小さいが、高速のメモリに移して利用すれば高速化を達成できる。これがキャッシュ・メモリ（以下キャッシュ）と呼ばれる方式である [2]。

主記憶とキャッシュの間はブロックを単位にデータの転送が行われる。プロセッサからのメモリアクセスが行われると、アクセスが行われたアドレスを含むブロックがキャッシュに存在するかが調べられる。キャッシュが主記憶のどの領域を保持しているかを記録しているのがタグである。タグを調べた結果、そのブロックが存在していると分かればデータアレイからプロセッサにデータを転送する。

しかし、主記憶共有型の並列計算機において各プロセッサがキャッシュを共有する場合には複数のキャッシュの内容を一致させる必要がある（一貫性制御）。

スヌープ方式で、一貫性制御が行われている場合を考える。一貫性制御の情報がネットワークを流れた時、一貫性制御の必要の有無は、プロセッサの各自のタグをアクセスしてはじめて分かる。しかし、プロセッサからも、キャッシュ・アクセスにともない、タグにアクセスが行われる。よって、この2つのアクセスが競合する可能性がある。

これを回避するためにキャッシュタグのコピーであるバックマップを用意する。ネットワークを流れた一貫性制御のためのアクセスは、バックマップに対して行うようにする。これにより、タグに対するアクセスの競合がなくなり、プロセッサからのキャッシュアクセスと一貫性制御が高速に行われる。

2.4 メモリアクセス・バッファリングを並列計算機に実現する際の問題

前節までに述べた、メモリアクセスのバッファリングを行い、かつキャッシュを備えた並列計算機を考える（図4）。

このモデルでは、システム内には複数のトランザクションが同時に存在し得る。この時ある2つのトランザクションが同一キャッシュブロックに対するトランザクションであった時、システム内の一貫性が保証できなくなる状況が発生する。具体的には、以下の2つの状況でシステム内の一貫性が保証できなくなる可能性がある。本稿では、これをトランザクションの競合と呼ぶ。

- メモリアクセス要求がバッファリングされている状況で、同一アドレスへのメモリアクセスがバス上で発生し、それにともなう一貫性制御により、キャッシュの状態が変更された場合
- あるアドレスへのメモリアクセスが生じ（要求パケットの送信）、そのアクセスが完了する（応答パケットの受信）前に同一のアドレスへのメモリアクセスが発生し、後者のアクセスに対する一貫性制御が前者のアクセスの完了を追い越した場合

前者の例を、図5に示す。2つのデバイスA,Bが同一ブロックのキャッシュコピーを持っていると仮定する。デバイスAで書き込みが発生し、それにともなう無効化要求をバッファリングしていた状況で、デバイスBからそのブロックに対する無効化要求が送出されたとすると（図5左上）、デバイスAのキャッシュが無効化された後にAの無効化要求が送出される。したがって、本来最終的にはAに最新のデータが存在していなければならないが（図5右）、Aには無効なブロックが残ったままになってしまう（図5左下）。

次に後者の具体的な例として、デバイスAが読み込み要求をだし、それをデバイスCが受け取った状況を考える(図6左上)。デバイスCがその応答をバスに送出する前に、デバイスBが無効化要求を送出し(図6右上)、この後デバイスCからの応答が流れる(図6左下)と、システムに矛盾が発生する。なぜなら、デバイスDの無効化要求は、デバイスAの読み込み要求の後にバスにながれたので、このブロックのキャッシュコピーはデバイスAには存在してはならない。しかし、デバイスAに応答が返ってくると、デバイスAにもこのキャッシュコピーが存在することになるからである。

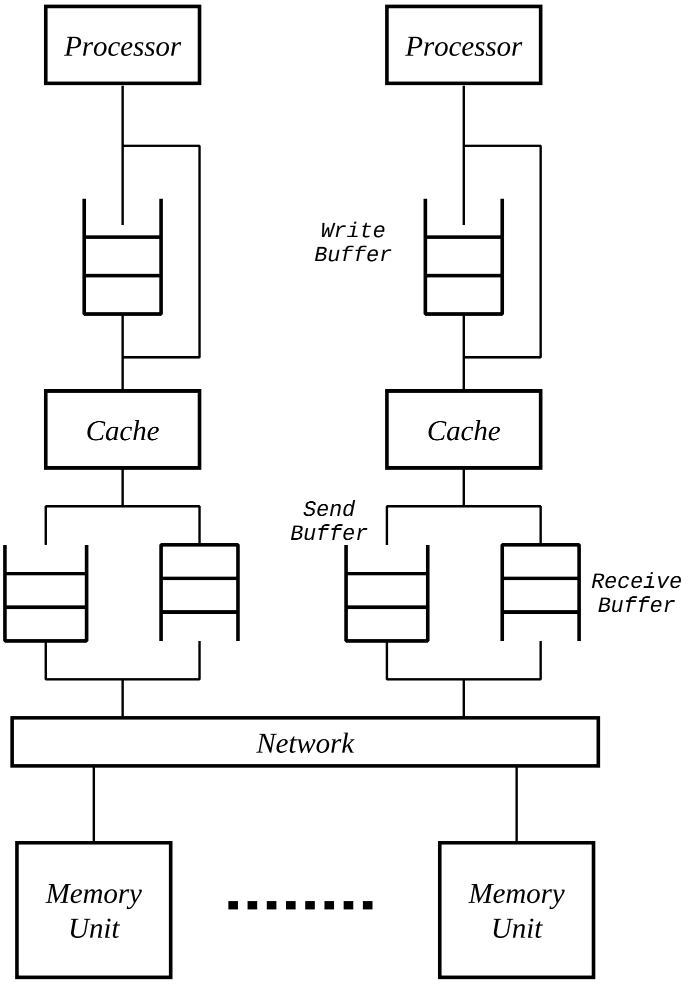


図4: 並列計算機モデル

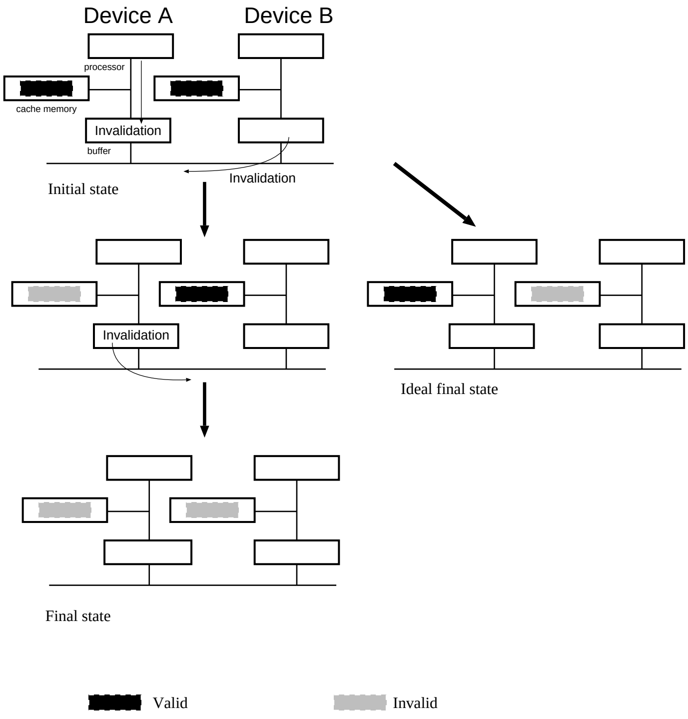


図5: 要求パケットがバッファリングされている時にキャッシュの状態が変わった状況

このような矛盾を生じさせないため、前者に対しては、バス上にながれたメモリアクセス要求によりキャッシュの状態が変更された場合には送信バッファ内の対応する要求を適切なものに変更する機構が、後者に対しては、このような状況が発生するのを禁止する機構がそれぞれ必要になる。

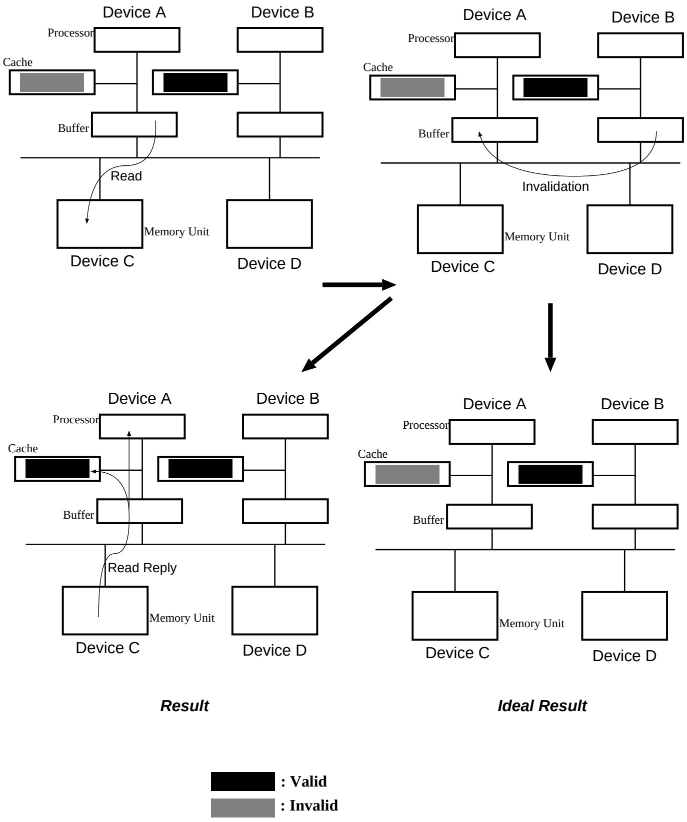


図 6: 応答パケットが返ってくる前に別のメモリアクセスによる一貫性制御が行なわれた状況

3 超並列計算機 JUMP-1 におけるメモリアクセス・バッファリングの実装

3.1 超並列計算機 JUMP-1 の概要 [3]

3.1.1 JUMP-1 クラスタの概要

JUMP-1 はクラスタ構造を採用している。各クラスタの構成を図7に示す。クラスタは、4つのプロセッサ・ユニット、2つのメモリ・ユニット、ならびに他クラスタとのインタフェースを司るルータ [3][4] で構成される。さらに、プロセッサ・ユニットは要素プロセッサ、2次キャッシュ、キャッシュ・コントローラ、およびバックマップの4つの部分からなる。また、メモリ・ユニットはクラスタメモリとそれを管理するメモリ・ベースド・プロセッサ(MBP)[3][5] で構成する。プロセッサ・ユニットとメモリ・ユニットは共有バスによって接続されている。この共有バスをクラスタバスと呼ぶ。

要素プロセッサとしては、メモリ管理ユニットおよび1次キャッシュを内蔵する市販の高性能 RISC プロセッサを用いる。1次キャッシュの仕様を表 1に示す。

JUMP-1 は分散共有メモリ型の並列計算機であり、主記憶は、各クラスタに分散して配置されている。しかし共有仮想記憶方式を利用し、他クラスタ上のメモリへのアクセスに対して、クラスタメモリを3次キャッシュとして動作させることで共有メモリ環境を提供する。

以下本章では、2章で述べたメモリアクセスの高速化手法をプロセッサ・ユニットへ実装する方法について述べる。キャッシュ・コヒーレンスはクラスタ内はスヌープ方式、クラスタ間はディレ

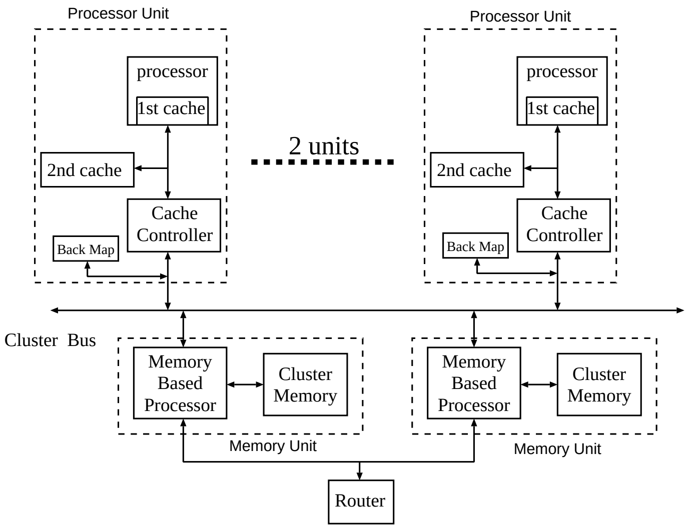


図 7: クラスタの構成

表 1: 1次キャッシュ

	命令	データ
アドレッシング	実アドレス	
メモリ更新方式	—	ライト・スルー
状態	Valid/Invalid	
コヒーレンス制御	Invalidation	
容量	20KB	16KB
ラインサイズ	64B	32B
ウェイ数	5	4
セット数	64	128
ライト・バッファ	—	8 Double Words

クトリ方式により行われる。2次キャッシュの仕様を表2に、また2次キャッシュの各状態を表3に示す。

表 2: 2次キャッシュの仕様

メモリ更新方式	ライト・バック/ライト・スルー
状態	Invalid, Local Shared Clean, Global Shared Clean, Exclusive Dirty, Local Shared Dirty
コヒーレンス制御	Invalidate/Update 選択可
容量	1MB
ラインサイズ	32B
ウェイ数	1（ダイレクトマップ）
セット数	32K
ライト・バッファ	8 Cache Blocks

表 3: 2次キャッシュの状態

状態	説明
Invalid	当該ブロックは無効化されている。
Exclusive Dirty	当該ブロックの最新データは、自デバイスのキャッシュにしか存在しない。
Local Shared Clean	当該ブロックはクラスタ内で共有されている可能性がある。内容は主記憶と等しい。
Local Shared Dirty	当該ブロックはクラスタ内で共有されている可能性がある。内容は主記憶とは異なる。
Global Shared Clean	当該ブロックはクラスタ間で共有されている可能性がある。内容は主記憶と等しい。

※ キャッシュコピーがクラスタ間で移動する時メモリにも書き戻しが行われる。よって、Global Shared Dirty という状態は存在しない。

3.1.2 プロセッサ・ユニットの構成

各プロセッサ・ユニットのブロック図を図8に示す。

キャッシュタグとデータアレイは、図7の2次キャッシュを構成するモジュールである。送信バッファ、受信バッファ、リプレース・バッファは、キャッシュ・コントローラを構成するモジュールである。送信バッファ、受信バッファは、それぞれクラスタバスに対するパケット送受信のためのモジュールである。リプレース・バッファについては3.2.2項でその働きを述べる。

3.2 プロセッサ・ユニットの動作 [8]

3.2.1 メモリアクセス処理

プロセッサ・ユニットにおいて処理されるメモリアクセスには、プロセッサ・ユニット内から生じたメモリアクセスと、プロセッサ・ユニット外から生じたメモリアクセスの2種類ある。それぞれのメモリアクセスは次のように処理される。

プロセッサ・ユニット内から生じたメモリアクセス

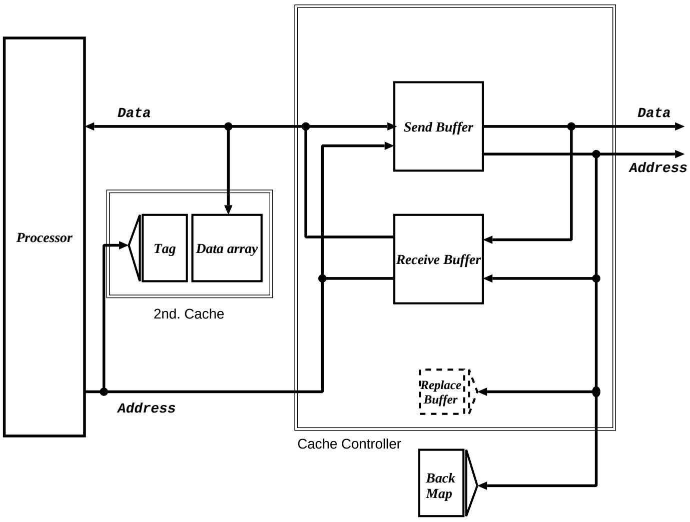


図 8: プロセッサ・ユニットのブロック図

(a) リード・アクセス まず、キャッシュタグにアクセスを行い、ヒットであればデータアレイからプロセッサに対してデータが渡される。ミスであれば、キャッシュブロックの読み込み要求のパケットが一旦バッファリングされ、その後クラスタバスに対して送出される。応答パケットが返ってくると一旦バッファリングされ、その後、プロセッサと2次キャッシュに渡される。

(b) ライト・アクセス プロセッサが発行したライト・アクセスのライトデータは当該アクセスで指定されたライト・アロケーション・ポリシに応じて、2次キャッシュのデータ・アレイ、あるいは送信バッファに書き込まれる。次に、必要に応じて、ライト・アクセスに対応する一貫性制御のためのパケットが生成、バッファリングされ、バスに送出される。

ライト・アロケーション・ポリシにはアロケート・ライト/ノー・アロケート・ライトの2種類ある。これは、2次キャッシュ・ミス時の動作が異なる。

- アロケート・ライト
  キャッシュ・ミスにともない、2次キャッシュにキャッシュ・ブロックをアロケートする。
  ライトデータは2次キャッシュ・ミスにも関わらず、直ちにデータ・アレイに書き込まれる。つまり、2次キャッシュを一種のライト・バッファとして用いている。しかしながら、JUMP-1 の2次キャッシュはライトミス時にはブロックの割り当てならびにブロックのフェッチを行うため、フェッチされてきたブロックを選択的にキャッシュへ書き込まなければならない。この選択、つまり、ブロック内のバイト単位にキャッシュ内のデータが最新か否かを示すモディファイド・ビットを設ける。
- ノー・アロケート・ライト
  キャッシュ・ミスの場合、2次キャッシュには、キャッシュブロックをアロケートしない。このため、他のデバイスのメモリに書き込みが行われる。

(c) ライト・バック キャッシュ・ブロックが置き換え対象となった時、その時の内容が主記憶と異なっていれば、書き戻しを行う。データ・アレイからデータを取り出して、送信バッファに一旦バッファリングされ、クラスタ・バスに送出される。

無効化要求のパケットと、読み込み要求のパケットを送出した場合、応答が返ってまで、送信パケットを送出したことを覚えておく必要がある。これは、3.2.2項で述べる、リトライを行うのに必要となるからである。このため、これらのパケットを送出した場合は、応答が返ってくるまでは送信バッファ内のエントリは解放されない。

また、読み込み要求のパケットを送出したとき、それがリード・アクセスによるものなのか、ライト・アクセスによるものなのかを区別しておく必要がある。なぜなら、リード時とライト時

では、応答パケットが返ってきてからの処理に違いがあるからである(プロセッサにデータを渡す / 渡さない)。

プロセッサ・ユニット外からのメモリアクセス 他のプロセッサ・ユニットの読み込み要求に対して、自プロセッサ・ユニットのキャッシュが最新のデータを所有していれば、自プロセッサ・ユニットがそれに対する応答を行わなければならない場合がある。

自プロセッサ・ユニットが応答を行う必要があるかどうかは、バック・マップに対してアクセスした結果によって判断される。応答する必要がある場合、データ・アレイからデータを読み出して、応答パケットを生成し、送信バッファにバッファリングされ、クラスタバスに送出される。

3.2.2 一貫性制御

2.4節で、メモリアクセス・バッファリング機構とキャッシュを備えた並列計算機においては、システムの一貫性が保証できなくなる場合があると述べた。本項では、JUMP-1 においてシステムの一貫性を保証するための方法について述べる。

プロセッサ・ユニット内の一貫性制御 2.3節で、キャッシュタグに対するアクセスを回避するために、そのコピーであるバックマップを設けると述べた。しかし、両者への書き込み制御の方法によっては一貫性が保証できない場合がある。

プロセッサの 2 次キャッシュアクセスにより、キャッシュブロックのリプレースが行われる場合を考える。この時、直ちに当該ブロックを無効にし、キャッシュタグにはその旨を書き込む。しかし、バックマップには直ちに書き込みを行うことはできない。なぜなら、バックマップへのアクセスの主導権はクラスタ・バス側にあり、クラスタ・バスの使用権を獲得した後でなければプロセッサはバックマップへアクセスすることができないからである。よって、キャッシュの状態に変更があってもバスを取得するまではバックマップに書き込みを行うことができない。したがって、キャッシュタグを変更した後、バックマップを変更するまでの間に、クラスタバスからそのブロックに対してアクセスが行われた時、データアレイにはそのブロックのデータが存在しないにも関わらず、バックマップは当該ブロックが存在すると判断してしまう可能性がある。

これを回避する手段として、次の方法が考えられる。

- リプレースを行う必要が生じた時、バスを取得できるまでリプレースを行わない。
- リプレースは直ちに行う。ただし、リプレースを行ったことを記録しておき、クラスタバスからバックマップにアクセスを行った時、その記録を用いてキャッシュタグとバックマップとの間の内容の違いを補正する。

前者の方法では、リプレースの必要が生じてから、実際にリプレースが行われるまでプロセッサがストールする。このプロセッサ・ストールを避けるため、JUMP-1 では後者の方法を採用している。このとき、リプレースされたブロックの情報を記録しておくのがリプレース・バッファである。あるデバイスにより、クラスタバスにパケットが送出された時、バックマップと同時にリプレース・バッファにもアクセスを行う。もしリプレース・バッファにヒットすれば、仮にバックマップにヒットしてもそのパケットはミスとして扱う。これにより、プロセッサ・ユニット内の一貫性制御が行われる。

プロセッサ・ユニット間の一貫性制御 2.4節で、トランザクションの競合によって、システム内の一貫性が保証できなくなると述べた。JUMP-1 におけるトランザクション競合の際の一貫性の保証法を以下に述べる。

要求がバッファリングされている際にキャッシュの状態が変更された際の、一貫性の保証法 2.4節で述べたように、要求パケットがバッファリングされている状況で、同一アドレスに対する別のトランザクションの要求パケットがバスに送出され、それにともない、自キャッシュの状態が変更された場合、バッファ内の要求パケットを適切なものに変更する必要がある。

JUMP-1 における、プロセッサからのアクセスの種類、キャッシュの状態と開始するトランザクションの関係を表 4に示す。

トランザクションが開始した後にキャッシュの状態が変化した場合、新しい状態に対応したトランザクションに変更される。

表4に示すように、このような状態が生ずるのは、ストア、および、アトミック・ロード・ストアにともなう要求を送信する場合である。

要求に対する応答の前に、同一アドレスへの一貫性制御が行われた際の、一貫性の保証法 2.4節で、あるアドレスへのメモリアクセスが発生し、そのアクセスが完了する前に同一アドレスへのメモリアクセスに対する一貫性制御が完了した場合、システム内の一貫性が保証できなくなると述べた。

JUMP-1 においてはこの際の一貫性を保証するためには次の方法を採る。すなわち、ある種のメモリアクセスが行われ、要求パケットがバスに送出され、それが完了する前 (応答パケットが返ってくる前) に、同一アドレスに対するメモリアクセスにともなう要求パケットがバスに送出された場合、後者のメモリアクセスの要求パケットを一旦キャンセルし、後でもう一度送出する。これをリトライと呼ぶ。

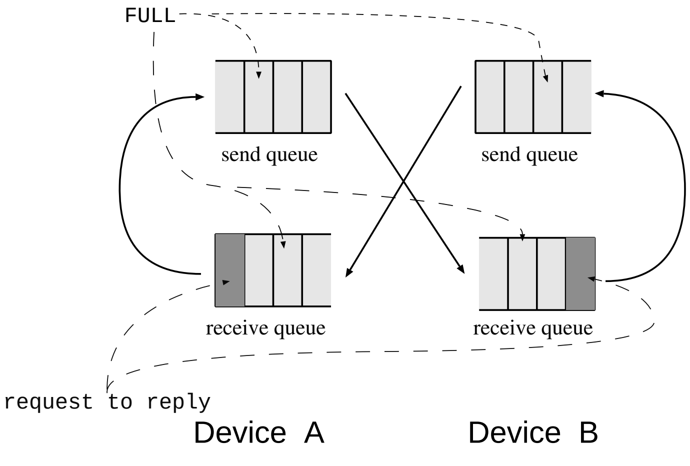
ただし、リトライを要請することで、デッドロックが発生する状況が存在する (例えば、無効化要求を送出した後に、他クラスタからの無効化要求がバスに送出された場合) ため、先行するトランザクションと後続トランザクションの関係で、リトライを行なうか否かを判定しなければならない。

表 5、および表 6に、JUMP-1 でリトライを要請する組合せを示す。これらの表では、要求パケットをバスに送出してその応答パケットが返ってくるまでに別の要求パケットが送出された場合、リトライを要求すべきか否かを示している。表 5は、後続の要求パケットの発行元がクラスタ内の場合を、表 6は後続の要求パケットの発行元がクラスタ外の場合である。

3.3 プロセッサ・ユニット内の各モジュールの構成 [6][7]

3.3.1 送受信バッファ

送受信のバッファはデッドロック回避のため (図 9参照)、それぞれ要求パケット用のバッファ、応答パケット用のバッファに分かれている (図 10)。さらに、要求送信用のバッファは論理的には 2 つに分かれている。それぞれ、プロセッサのロードに対応したパケット、ストアに対応したパケットを処理する。これは、ロード時のプロセッサ・ストールを可能な限り短くするためである。



※ デバイス A、B がそれぞれお互いに対して要求を行おうとしているが相手の受信バッファに空きがなく、さらに受信バッファの先頭には応答を行なわなければならない要求があるが、送信バッファも空きがないため待たされている状況を示している。これを回避するには、送受信のキューを要求と応答で別々にし、応答のパケットが存在する時は、要求に先だってバスに送出するようにすればよい。

図 9: デッドロックが発生している状況

表 4: プロセッサからのアクセスと対応するトランザクション

プロセッサからのアクセス				キャッシュコピーの状態	トランザクション
ロード / ストア	キャッシュ制御の方針				
	CC	WA	CP		
ロード	NC	—	—	—	*Nn-Cchbl. Read*
	C	—	—	Invalid	*Shared Read*
				Otherwise	—
ストア	NC	—	—	—	*Nn-Cchbl. Write*
	C	A	I	Invalid	*Exclusive Read*
				Exclusive Dirty	—
				Otherwise	*Invalidation*
			U	Invalid	*Shared Read*
				Exclusive Dirty	—
				Otherwise	*Update*
		NA	I	Invalid	*No-Alloc. Write*
				Exclusive Dirty	—
				Otherwise	*Invalidation*
			U	Invalid	*No-Alloc. Write*
				Exclusive Dirty	—
				Otherwise	*Update*
ロード・ストア	NC	—	—	—	*Nn-Cchbl. Swap*
	C	—	I	Invalid	*Exclusive Read*
				Exclusive Dirty	—
				Otherwise	*Invalidation*
			U	Exclusive Dirty	—
				Local Shared *	*Update*
				Otherwise	*Swap*

CC: キャッシャビリティ
C: キャッシャブル
NC: ノンキャッシャブル

WA: ライト・アロケーション
A: アロケート
NA: ノーアロケート

CP: コンシステンシ・ポリシ
I: ライト・インバリデート
U: ライト・アップデート

表 5: リトライを行う状況（クラスタ内）

先行パケット	後続パケット				
	Shared Read	*Exclusive Read*	*Invalidation*	*Update*	*Swap*
Shared Read	○	○	○	○	○
Exclusive Read	○	○	ϕ	ϕ	○
Invalidation	○	○	ϕ	ϕ	○
Update	×	×	×	×	×
Swap	○	○	○	○	○
Write Back	×	×	×	×	×

○：リトライを行なう
×：リトライを行なわない
ϕ：起こり得ない.

各要求の意味

Shared Read	読み込み要求
Exclusive Read	読み込み要求兼無効化要求
Invalidation	無効化要求
Update	更新要求
Swap	内容を交換する要求。プロセッサからの不可分なロード・ストアに対応して発行される。
Write Back	キャッシュの中身をメモリに書き戻す要求

表 6: リトライを行う状況（クラスタ間）

先行パケット	後続パケット				
	Shared Read	*Exclusive Read*	*Invali- dation*	*Update*	*Locali- zation*
Shared Read	○	○	○	○	○
Exclusive Read	○	○	○	○	○
Invali- dation	○	○	×	×	ϕ
Update	ϕ	ϕ	×	×	×
Swap	○	○	○	○	○
Write Back	ϕ	ϕ	ϕ	ϕ	ϕ

○ : リトライを行なう
× : リトライを行なわない
ϕ : 起こり得ない.

各要求の意味（表 5に示した要求は省略）

Localization キャッシュブロックのクラスタ間共有状態を Local にする。

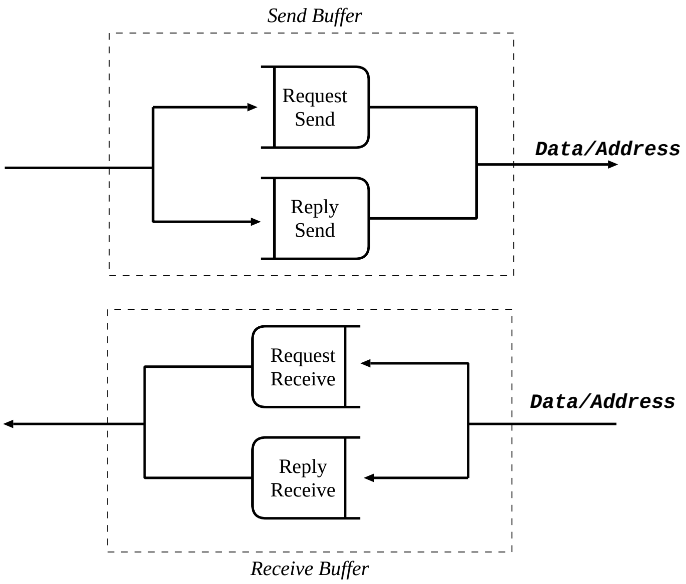


図 10: 送受信バッファの内部構成（概要）

実際の構成では、図 11に示すように、データやアドレスを保持するバッファは共有されており、パケット送出の制御部だけが各バッファごとに個別に設けられている。共有バッファのエントリ数は 8 である。また、ストア時の読み込み要求を行う時などに必要となるモディファイド・ビットが各エントリに対して備えられている。また、2.4節で述べたように、クラスタバスに同一アドレスに対するパケットがながれた場合、バッファリングされているパケットの種類を変更しなければならない場合がある。このため、各エントリには、クラスタバスにながれたパケットのアドレスと比較を行うために比較器が備えられている。

また、3.2.2項で述べたリトライを実装するには、*Shared Read*,*Exclusive Read*,*Invalidation*,*Swap* の各パケットの送出を行ったことを、応答が返ってくるまで覚えておく必要がある。よって、各エントリには、そのパケットを送出済みでかつ応答パケットを待っている状態を示すビットを設ける。

また、受信用バッファの構成を図 12に示す。エントリ数は 2 である。送信バッファと同様、アドレス、データなどの情報を格納するバッファは共用し、制御部のみを要求系と応答系で個別に設ける。

3.3.2 リプレース・バッファ

ハードウェア構成を図 13に示す。JUMP-1 で、プロセッサからキャッシュの状態を書き換えるのは、キャッシュブロックをリプレースした場合だけである。よって、リプレース・バッファには、状態を書き換えた（すなわち無効化した）アドレスだけを記録する。バッファのエントリ数は 4 である。

リプレース・バッファの動作を以下に示す。

1. エントリの書き込み
   キャッシュブロックがリプレースされた場合、空いているエントリにブロックアドレスを書き込む。空いているエントリがなければ、プロセッサをストールさせる。

2. バスをながれたパケットのアドレスとの比較
   バスにパケットがながれた時、パケットのブロックアドレスに一致するエントリがあるか調べる。

3. エントリの解放
   送出されたパケットのブロックアドレスが記録されているブロックアドレスと一致すると、そのエントリを解放する。同時にバックマップのブロックを無効化する。

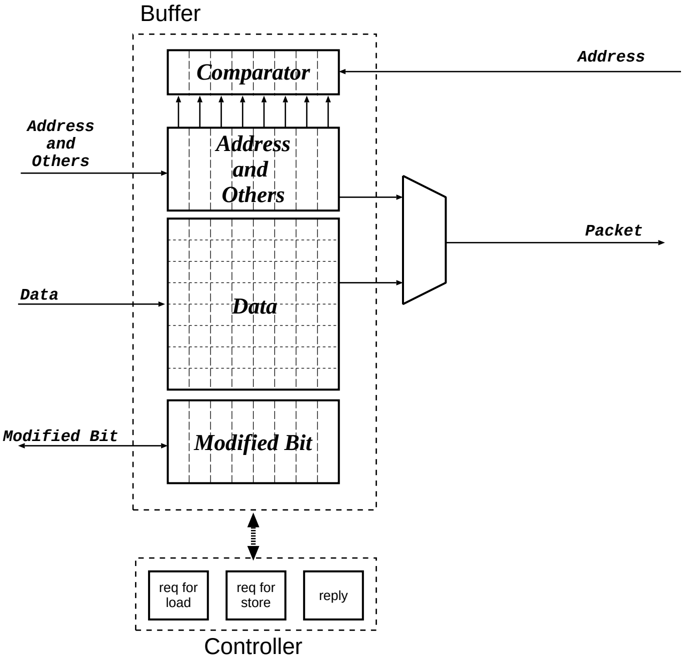


図 11: 送信部の内部構成

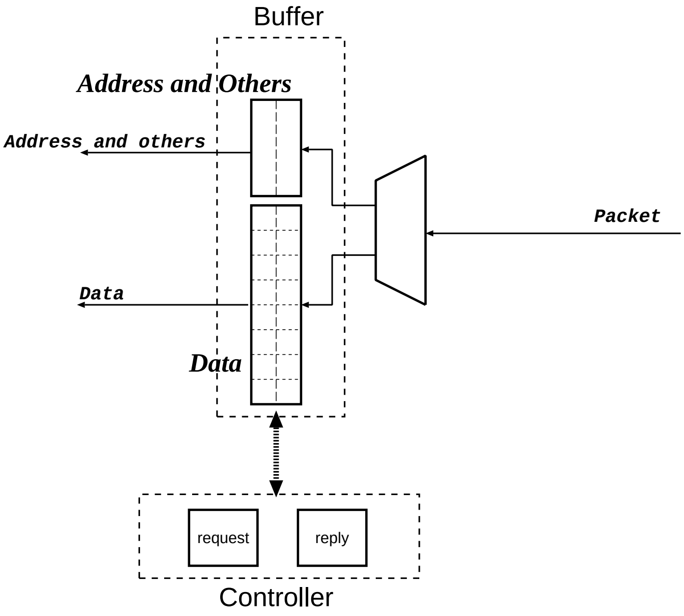


図 12: 受信部の内部構成

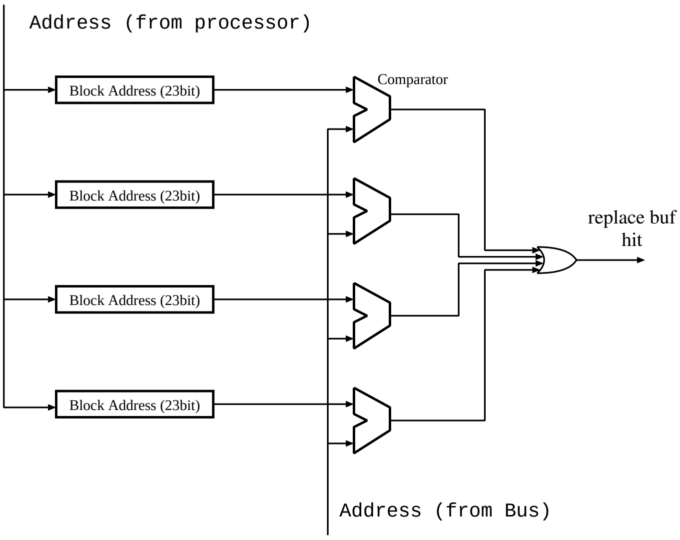


図 13: リプレース・バッファの構成

4 JUMP-1 のプロセッサ・ユニットの実装に関する考察

4.1 ハードウェア量に関する考察

プロセッサ・ユニットの実装に必要なハードウェアの考察を行う。

キャッシュ・コントローラに必要なゲート数

パケット送受信部　送受信部バッファに必要となるゲートの数は表 7の通り。全体では約 11,000 ゲートとなる。

ただし、ゲート数の関係上、送受信バッファのすべてを 1 つのチップに実装することはできない。よって、データ部だけは別のチップ (バッファ·チップ) に実装する。キャッシュ·コントローラ·チップに実装する部分を合計すると約 4,200 ゲートとなる。

リプレース·バッファ　リプレース·バッファを構成しているハードウェアは、23 ビットのレジスタと比較器がそれぞれ 4 つである。これに必要なゲート数は 500 ゲート程度である。

プロセッサ·ユニットに必要なハードウェア　プロセッサ·ユニットの各構成要素とそれに使用するハードウェア、および数量を表 8に示す。

4.2 メモリアクセス時間に関する効果

本稿ではメモリアクセス·レイテンシ短縮の技術のうち、メモリアクセスと演算のオーバラップ、メモリアクセス·パイプライニング、キャッシュメモリについて触れた。本節では、これらの高速化手法により、メモリアクセス時間にどのような効果が得られるかを示す。ただし、次のような仮定を設ける。

1. バス上での競合はない。
2. MBP(図 7参照)に要求パケットが到着してから、MBP から応答パケットが送出される時間(以下、主記憶アクセス時間)は N システムサイクル

4.2.1 キャッシュの効果

プロセッサからロードの要求が生じたとする。以下で、ロードか完了するまでの時間について述べる。

1 次キャッシュは、プロセッサに内蔵されている。1 次キャッシュにヒットした場合は、アクセスした次のサイクルでデータを得ることができる。

表 7: キャッシュ・コントローラの各バッファに必要なゲート数

送信部用

基本回路	1つ当たりのゲート数	用いる数量	計
レジスタ	2.25	アドレス:28×8 その他*: 約 50×8	約 1,400
比較器(Ex-or)	3	23×8	約 550
8-to-1 マルチプレクサ	15	28+50	約 1,200
メモリ (モディファイドビット用)	720	1	720
メモリ (データ用)	4752	1	4752

*:パケットの種別など

受信部用

基本回路	1つ当たりのゲート数	用いる数量	計
レジスタ	2.25	アドレス:25×2 その他:約 30×2	約 250
2-to-1 マルチプレクサ	4	55	220
メモリ (データ用)	2160	1	2160

表 8: プロセッサ・ユニットに必要なハードウェア

構成要素	使用するハードウェア	数量
プロセッサ	Super-Sparc +	1
2次キャッシュ・データアレイ	128K ×8 bits Synchronous SRAM	8
2次キャッシュ・タグアレイ	32K ×8 bits SRAM	4
2次キャッシュ・バックマップ	128K ×8 bits SRAM	2
キャッシュ・コントローラ・チップ	カスタム LSI 0.5μm ゲートアレイ	1
バッファ・チップ	カスタム LSI 0.5μm ゲートアレイ	1

1次キャッシュにミスした場合、アクセスした次のサイクルで、プロセッサの外にロードの要求が出される。2次キャッシュにヒットした場合は、この2サイクル後にブロック転送が始まる。クリティカル・ワード・ファースト方式[10]により、アクセスを行ったアドレスのデータを最初にプロセッサに渡すので、転送が行われた最初のサイクルにおいて、プロセッサの動作を再開することができる。

次に2次キャッシュにもミスした場合を考える。この場合は、主記憶に対するアクセスを行う必要がある。まずキャッシュ・コントローラから読み込み要求を送信し、それに対する応答が MBP から送られてくる。

この間の一連の処理の流れを以下に示す（アクセス経路を図14に、タイミングチャートを図15にそれぞれ示す。）。

A. ロード発生。

B. 1次キャッシュ・ミスを検知。2次キャッシュにアクセスを開始する。

C. 2次キャッシュ・ミスを検知。および、キャッシュ・コントローラに要求パケットのバッファリング

D. アービトレーション。キャッシュコントローラ内には、複数のパケットが存在するので、それらのうちどれを送出するのかを決める必要がある。たとえ、パケットが全くバッファリングされていなかったとしても、1サイクルかかる。

E. バス取得要求の送出。

F. バス取得が認められる。遅延時間や構成の関係上、キャッシュ・コントローラは、バス取得を要求してから、それが認められたことを認識するまで最低2サイクルかかる。

G. キャッシュ・コントローラから要求パケットが送出される。

H. MBP にパケットが届く。

I. 主記憶アクセス。

J. MBP から応答パケットが送出される。

K. キャッシュコントローラに届く。

L. 受信バッファへの書き込み。パケットの第1フリットは、アドレスなどの制御情報が記録されている。実際のデータは第2フリット以降に記録される。

M. プロセッサへの転送。受信バッファにデータが揃う前にデータの転送が開始される。クリティカル・ワード・ファースト方式により、転送が開始した最初のサイクルからプロセッサの動作を再開できる。

以上で $15+N$ サイクルかかる。

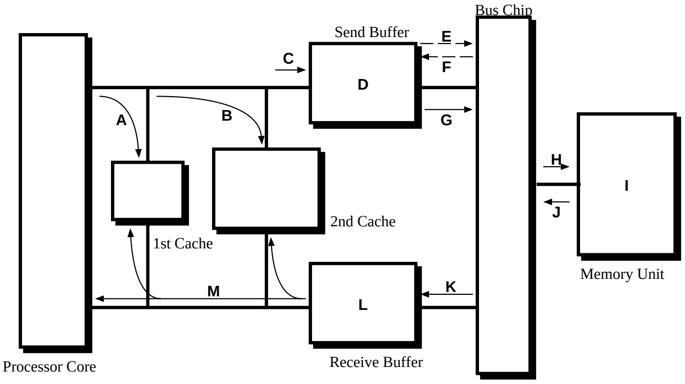


図 14: 2次キャッシュ・ミス時のアクセス経路

今、$N=5$ と仮定する。これは、主記憶アクセスにかかる時間がほぼ最短である場合の数字である。

キャッシュが存在しない時、プロセッサからのロードの処理に、17 サイクル（20 サイクル $-B-C$）かかる。

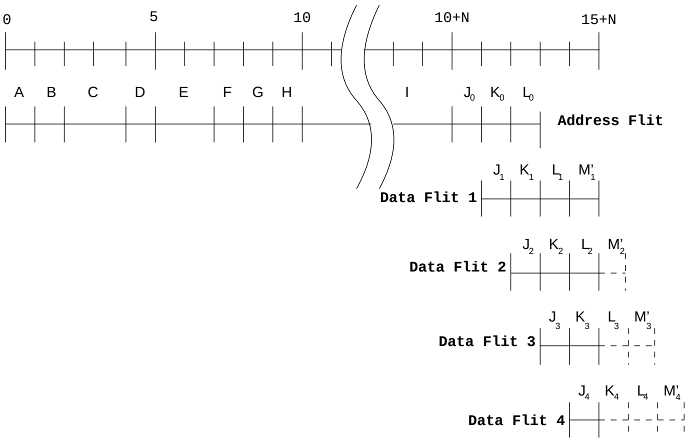


図 15: 2 次キャッシュ・ミス時のタイミングチャート

1 次キャッシュのみが存在する場合、1 次キャッシュヒット時で 1 サイクル、ミス時で 18 サイクル$(20$ サイクル $-\ C)$かかる。

よって、主記憶アクセス時間の平均が 5 サイクルであるとして、キャッシュヒット率を 95% とすると、平均アクセス時間は、

$$0.95 \times 1 + (1 - 0.95) \times 18 = 1.85$$

サイクルとなる。キャッシュが存在しない場合に比べて、約 11% の時間で済む。

1 次、2 次キャッシュの両方が存在する時は、1 次キャッシュヒット時で 1 サイクル、1 次キャッシュミス- 2 次キャッシュヒット時で 4 サイクル、1 次キャッシュミス- 2 次キャッシュミス時で 20 サイクルかかる。キャッシュヒット率をともに 95% とすると、平均アクセス時間は

$$0.95 \times 1 + (1 - 0.95) \times \{0.95 \times 4 + (1 - 0.95) \times 20\} = 1.19$$

サイクルとなる。

キャッシュが両方とも存在しない場合に比べて 7%、1 次キャッシュだけの時と比べて約 64% の時間で済む。

キャッシュメモリの効果は、Nの値が大きいと、さらに顕著になる。上と同じ条件の下で、$N = 500$ とすると、1 次、2 次キャッシュの両方が存在する場合の平均アクセス時間は、両方とも存在しない場合の約 0.46% の時間で済む計算になる。

4.2.2 ライトバッファの効果

本項では、メモリアクセスと演算の並列実行を行う効果について考察する。

2 次キャッシュへのアロケート・ライトがミスした場合を考える。並列実行を行う場合、データはヒットの場合と同様に、2 次キャッシュに書き込まれる。この場合、プロセッサからライトが発行されてから 5 サイクルで書き込みが終了する。

並列実行を行わない場合を考える。メモリアクセスにかかる時間は図 15の時よりも、応答パケット全てを 2 次キャッシュに書き込まなければならない分だけ長くなる。JUMP-1 では、キャッシュの 1 ブロックは 4 ワードである。1 ワードの転送には 1 サイクルかかる。つまり、メモリアクセスにかかる時間は、図 15の場合よりも、3 サイクル長くなる。つまり、$18 + N$サイクルである。

2 次キャッシュ・ヒットの確率を 95% とする。また、2 次キャッシュ・ライトがミスした時に、クラスタ内のメモリにヒットする確率を 95% として、クラスタ内メモリにヒットした場合の主記憶アクセス時間に 5 サイクル、ミスした場合の主記憶アクセス時間に 500 サイクルかかるとする。

メモリアクセスと演算の並列実行を行わない場合、アロケート・ライトにかかる平均時間は、

$$\begin{aligned} 0.95 \times 5 \quad &+ \quad (1 - 0.95) \times 0.95 \times 23 \\ &+ \quad (1 - 0.95) \times (1 - 0.95) \times 518 = 7.1375 \end{aligned}$$

サイクルである。並列実行を行った場合、常に 5 サイクルで済む。つまり、この仮定においては、約 29% の短縮が行われたことになる。

5 おわりに

本稿では、メモリアクセス・バッファリングとキャッシュメモリによるメモリアクセスのレイテンシ隠蔽技法と、それらの JUMP-1 における実装について述べた。

メモリアクセス・バッファリングを行うことで、システム内に一時的に一貫性が保たれない状況が発生するので、このような一時的な一貫性の欠如を回避あるいは保証するための制御も要求される。

4.2節では、メモリアクセス時間に関する考察を行った。この節で示したように、キャッシュを実装することによってメモリアクセスのレイテンシは、しない場合と比較して 7% になる。また、メモリアクセスと演算の同時実行を行うことによって、2 次キャッシュ・アロケート・ライトにかかる平均時間は、29% 短縮できる。

本稿においては、メモリアクセス・パイプライニングの具体的な効果については触れていない。現在 JUMP-1 は設計段階にあり、厳密な性能評価が今後の課題である。

謝辞

本研究の機会を与えてくださり，適切な御指導を賜わった富田眞治教授に深甚な謝意を表します。

　また，貴重なご助言をいただいた中島浩助教授，森眞一郎助手に深く感謝致します。

　さらに，共同研究者である五島正裕氏をはじめとして，日頃から御鞭撻下さった京都大学工学部情報工学科富田研究室の諸兄に感謝致します。

参考文献

[1] John L.Hennessy,David A.Patterson（富田 眞治、村上 和彰、新實 治男 訳）：「コンピュータ・アーキテクチャ」日経 BP 社. 1992 年 12 月.

[2] 富田 眞治、村上 和彰:「計算機システム工学」昭晃堂. 1992 年 4 月

[3] 「要素プロセッサと 2 次キャッシュ」文部省重点領域研究. 超並列原理に基づく情報処理基本体系. 第 3 回シンポジウム予稿集.pp.217-234

[4] 「相互結合網とルータ」文部省重点領域研究. 超並列原理に基づく情報処理基本体系. 第 3 回シンポジウム予稿集.pp.257-279

[5] 「MBP コアの基本構想」文部省重点領域研究. 超並列原理に基づく情報処理基本体系. 第 3 回シンポジウム予稿集.pp.235-244

[6] M.Goshima:「JUMP-1CC 内部仕様書」1994 年 1 月.

[7] M.Goshima, K.Hiraki:「JUMP-1 共通バス仕様書」1994 年 10 月.

[8] M.Goshima:「JUMP-1CBus 仕様書」1994 年 2 月

[9] Michel Dubois,Christoph Scheurich,Faye Briggs: "MEMORY ACCESS BUFFERING IN MULTIPROCESSORS " Proc.13th Int'l. Symp. Computer Architecture,pp.434-442,June 1986.

[10] 相越 克久、田中 長光 訳 多田 好克 監訳：「SPARC アーキテクチャ・マニュアル バージョン８」SPARC International,Inc. 1992 年 11 月.

付録 :JUMP-1 のパケットの種類

Shared Read Request メモリからの読みだし要求。

Exclusive Read Request メモリからの読みだし要求兼無効化要求

Write Back Request キャッシュブロックのリプレースに伴う、メモリへの書き戻し要求。

No-Allocate Write Request 他のデバイスのキャッシュコピーの内容を更新する。ただし、状態は更新しない。

Self Clean up Request キャッシュのダーティブロックの自発的なメモリへの書き戻し要求。このリクエストが発行されると、当該キャッシュブロックの状態はどのデバイスも *Local Shared Clean* となる。

Swap Request メモリへのスワップ要求。中身を交換する。

Update Direct Request キャッシュ、及びメモリへの更新要求

Update via Home Request この要求は、MBP が受け取る。その後 MBP が Update Direct Request を送出する。Update を明確にシリアライズすることができる。

Invalidation Direct Request キャッシュに対する無効化要求

Invalidation via Home Request この要求は MBP が受け取りその後 MBP が Invalidation Direct Request を送出する。

Flush Request キャッシュメモリをフラッシュするリクエスト

Localize Request 対象ページのクラスタ間共有状態を Local にする。

Read Reply Shared Read Request に対する応答

Exclusive Read Reply Exclusive Read Request に対する応答

Write Acknowledge 書き込み要求などの操作が終了したことを示す。

Nop 何もしない。出しかけの要求を取り消す時などに使われる。